LiteOS 移植

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本范例基于 BSP 中 GPIO_OutputInput 参考程序，介绍如何移植 LiteOS 并创建 2 个线程。
BSP 版本	M031_Series_BSP_CMSIS_V3.07.000
开发平台	NuMaker-M031SE3AE V1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本范例基于 BSP 中 GPIO_OutputInput 参考程序，介绍如何在 Keil MDK 编译环境下移植 LiteOS 并创建 2 个线程。线程 1 通过串口，每 1000 ms 输出“This is task1,count is ***”；线程 2 通过串口，每 2000 ms 输出“This is task2,count is ***”。

1.1 原理

LiteOS 是华为开发的一款轻量级操作系统，主要面向物联网（IoT）设备。特点：

- 体积小巧，内存占用低（最小内核可小于 10 KB）；
- 实时性强，适合对响应速度有要求的场景；
- 支持多种主流芯片架构（如 ARM Cortex-M、RISC-V 等）；
- 支持多种网络协议（如 Wi-Fi、NB-IoT、LoRa 等）；
- 代码开源（部分版本），易于二次开发和定制。

LiteOS 与 FreeRTOS、RT-Thread、Zephyr 等其他物联网 RTOS 类似，但在国产化、生态兼容性和与华为 IoT 生态的集成方面有独特优势，广泛应用于智能家居、个人穿戴、车联网、城市公共服务、制造业等领域，大幅降低设备布置及维护成本，有效降低开发门槛、缩短开发周期。

本范例使用 M031SE3AE 芯片，内置 128 KB Flash，16 KB RAM，介绍如何在 Keil MDK 中移植 LiteOS 并创建管理 2 个线程。

1.2 执行结果

本范例代码执行编译和烧录后，串口打印信息如图 1-1 所示。

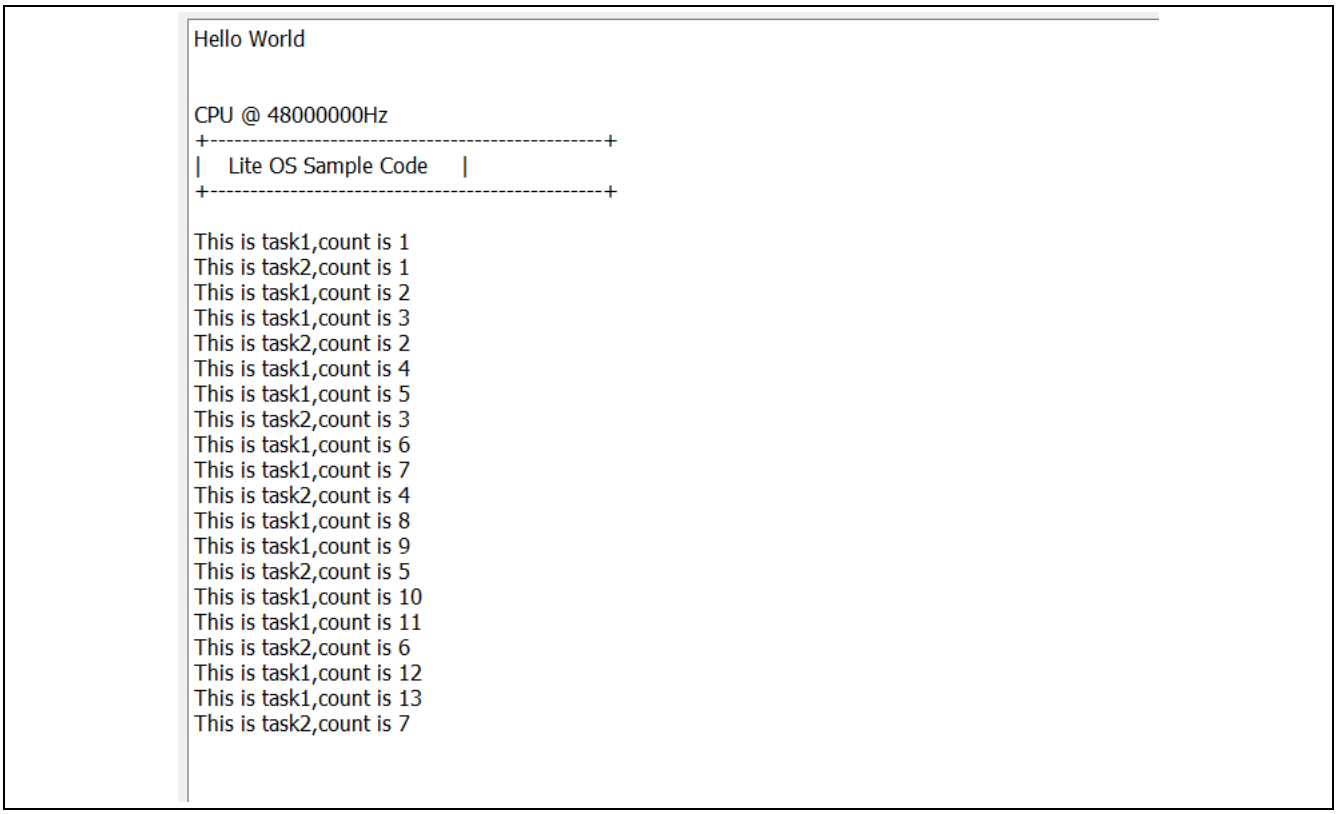

图 1-1 串口打印消息

2. 代码描述

2.1 下载 LiteOS 源码

下载 LiteOS 源码：

- 使用 develop 分支，链接为：https://github.com/LiteOS/LiteOS/tree/develop；
- 使用 LiteOSV200R001C50B038 标签，链接为：https://github.com/LiteOS/LiteOS/tree/LiteOSV200R001C50B038 。

以上两个链接均可使用，本范例中使用了 develop 分支，源码解压后如图 2-1 所示。

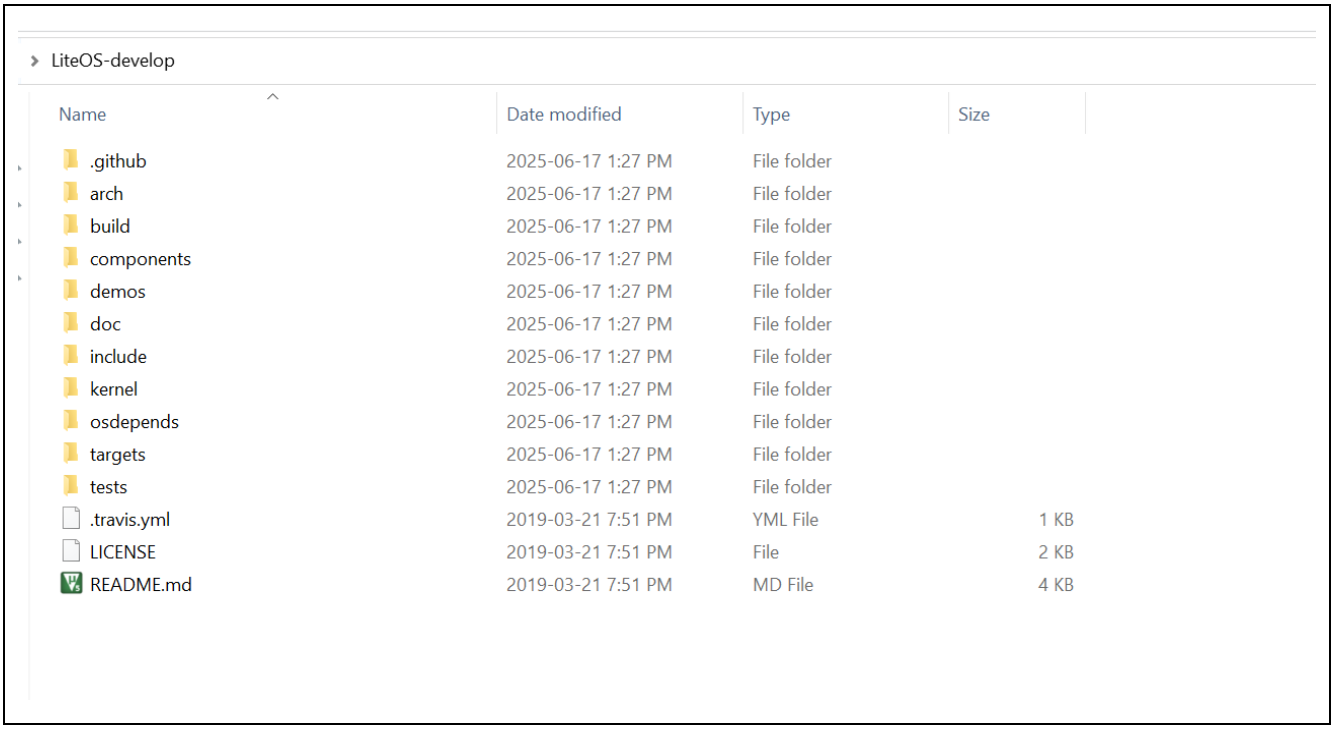

图 2-1 LiteOS 源码文件结构

2.2 拷贝 LiteOS 源文件到 BSP 路径下

先在 BSP\ThirdParty 目录下，建一个子目录 LiteOS-develop，用来放置 LiteOS 的源码。在新建 *LiteOS-develop* 目录下，再建四个子目录，分别叫 *kernel* 、*arch* 、*cmsis* 、*OS_CONFIG* ，如图 2-2 所示。

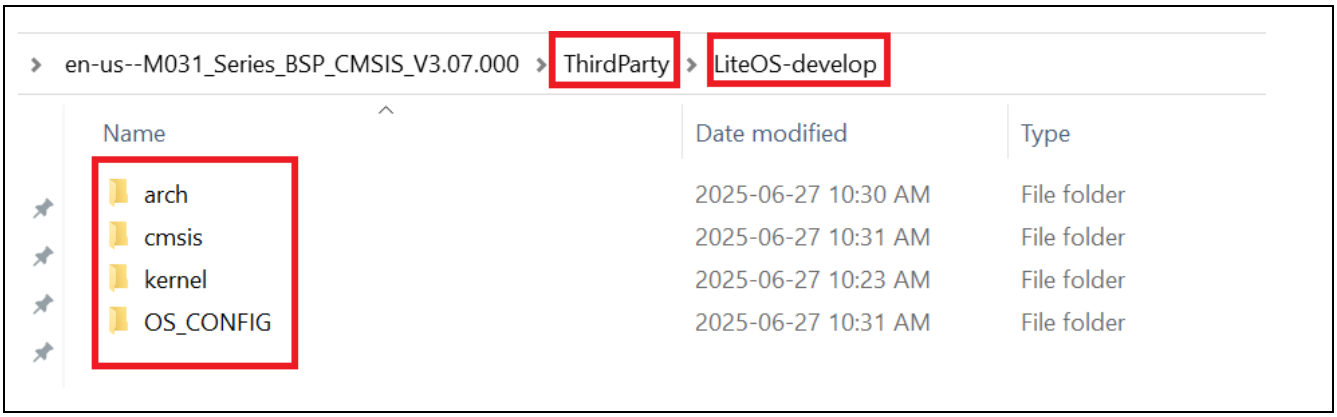

图 2-2 LiteOS 源码存放路径

把 LiteOS 源码.*kernel* 目录下的内容全部复制到工程里的 *LiteOS-develop\kernel* 目录下，然后再把.\LiteOS-develop\Kernel\base\mem 目录下的两个子目录 *bestfit*、*tlsf* 删除，如图 2-3 所示。

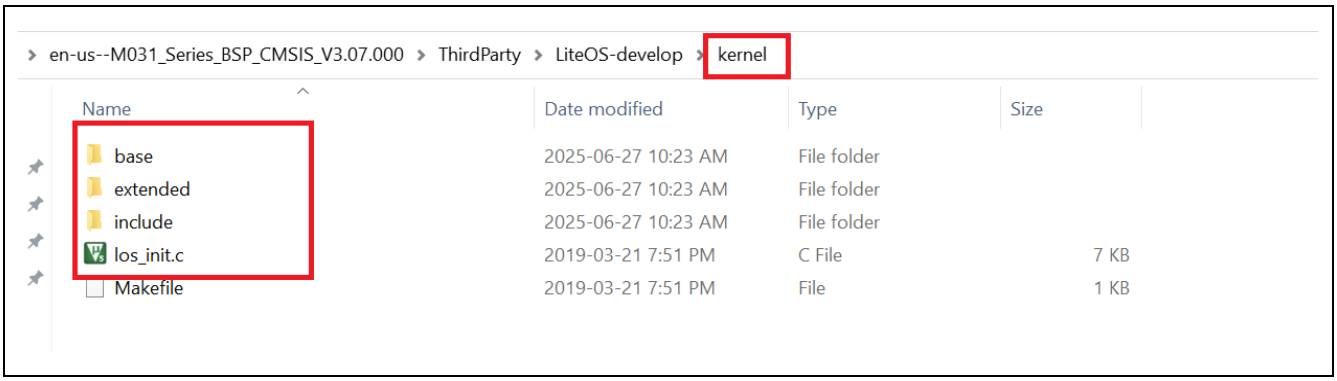

图 2-3 更改后 kernel 目录下文件结构

从 LiteOS 源码.*arch\arm\arm-m* 中复制 *src*、*include* 两个目录到工程 *LiteOS-develop\arch* 目录下。再复制 *arch\arm\arm-m\contex-m0\keil\los_dispatch_keil.S* 到工程 *LiteOS-develop\arch* 目录下，如图 2-4 所示。

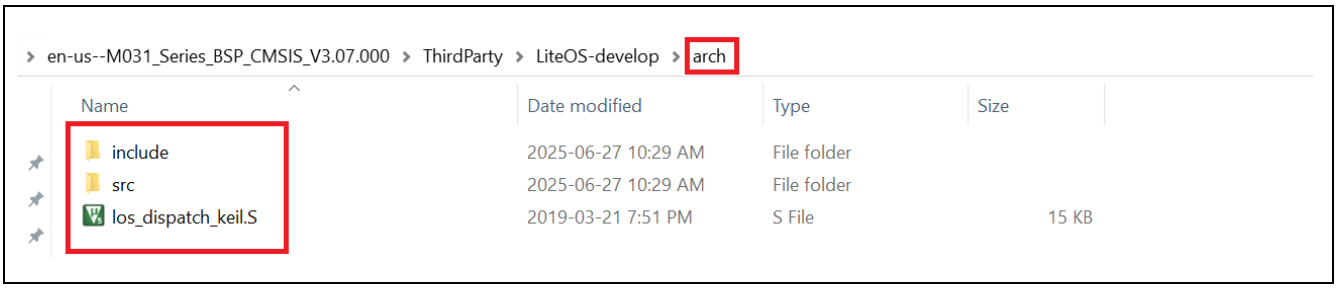

图 2-4 更改后 arch 目录下文件结构

复制 LiteOS 源码 *osdepends\liteos\cmsis* 目录中的文件到工程 *LiteOS-develop\cmsis* 目录下，如图 2-5 所示。

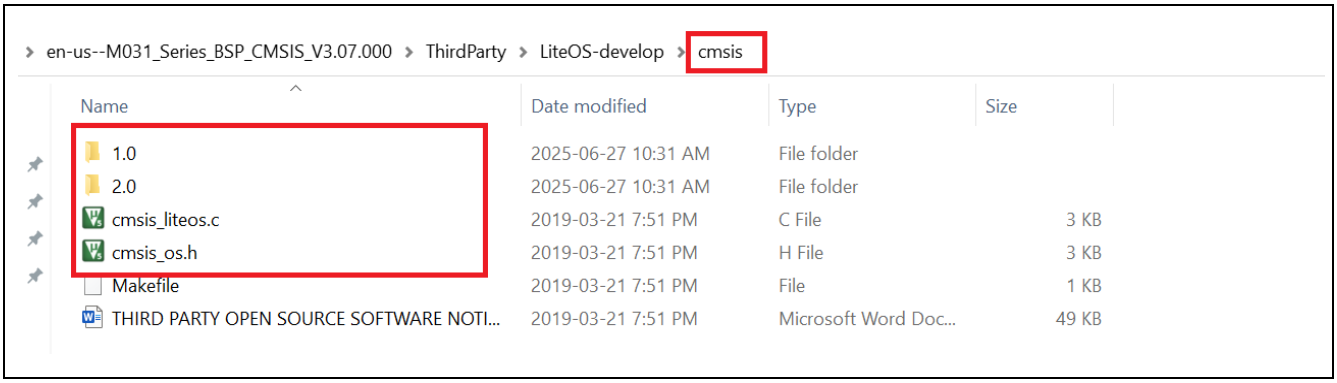

图 2-5 更改后 cmsis 目录下文件结构

复制 LiteOS 源码 *targets\STM32F103VET6_NB_GCC\OS_CONFIG* 目录下头文件（.h 文件）到工程 *LiteOS\OS_CONFIG* 目录下，如图 2-6 所示。

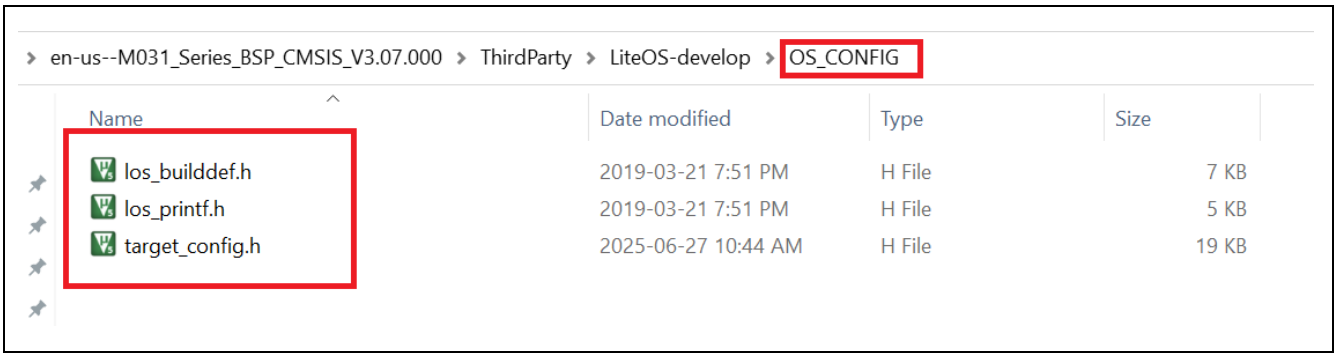

图 2-6 更改后 OS_CONFIG 目录下文件结构

到此，移植 LiteOS 所需的文件就准备好了。

2.3 建立 LiteOS Keil project

打开在 BSP 包中 GPIO_OutputInput 工程，将其名称修改为 LiteOS，如图 2-7、图 2-8 所示。

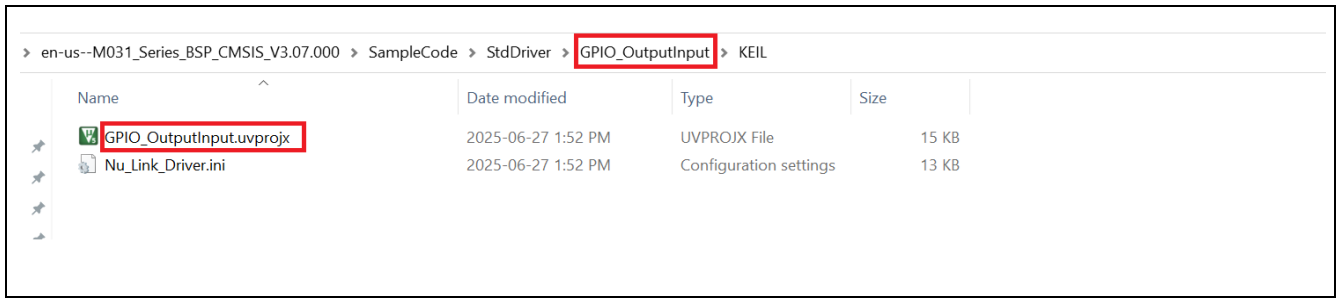

图 2-7 修改前为 GPIO_OutputInput 工程目录

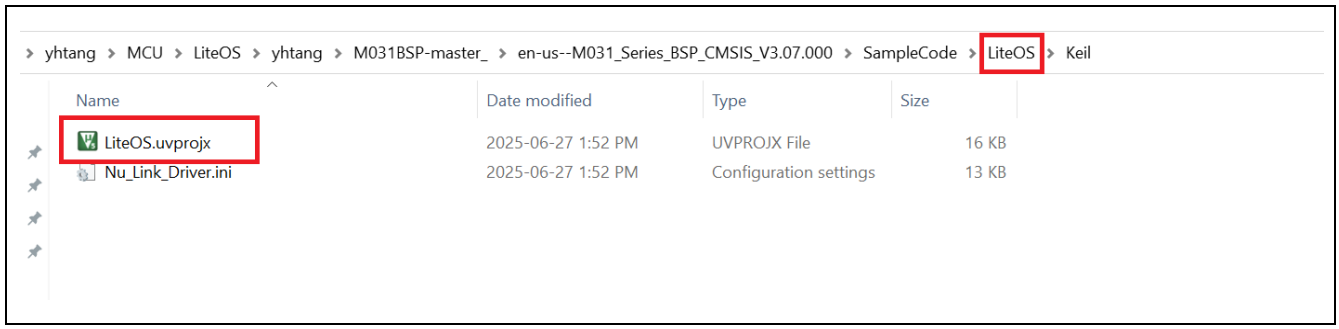

图 2-8 修改后为 LiteOS 工程目录

打开 LiteOS 工程文件，修改 Project Target 为 LiteOS，并创建 *LiteOS/Arch* 、*LiteOS/Kernel* 、*LiteOS/CMSIS LiteOS/Config* 四个 Group，如图 2-9 所示。

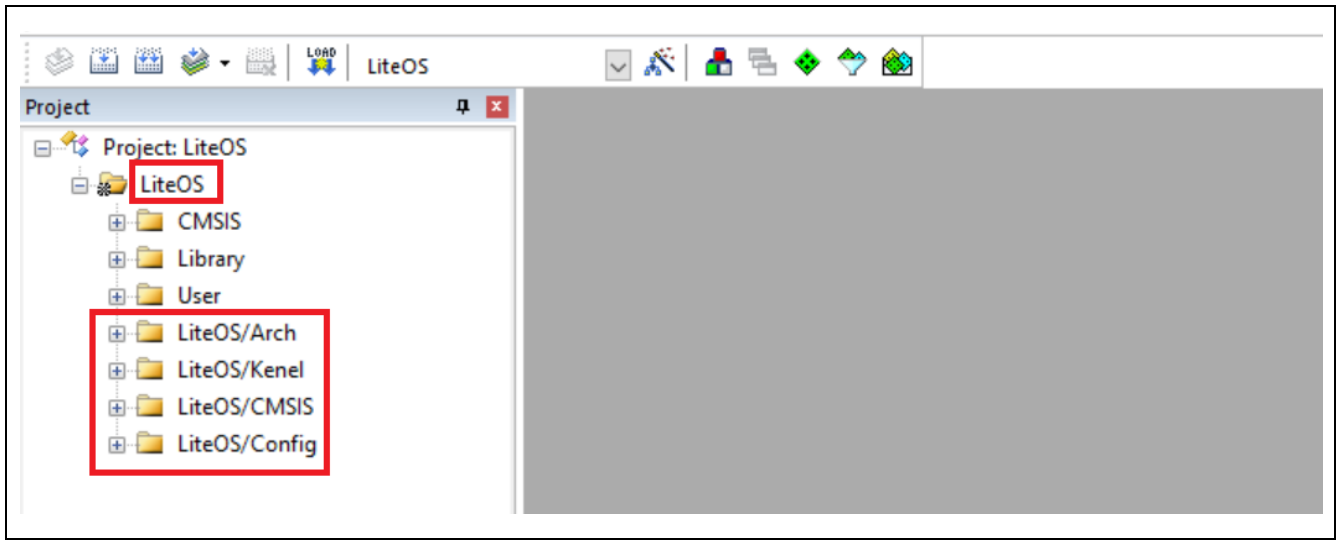

图 2-9 LiteOS Project Items

在 *LiteOS/Kernel* 分组下，添加 *LiteOS\Kernel* 目录下（包含子目录下）的所有.c 文件。

在 *LiteOS/Arch* 分组下，添加 *LiteOS\Arch* 目录下（包含子目录下）所有.c 文件，以及 *los_dispatch_keil.S* 文件。

在 *LiteOS/CMSIS* 分组下，添加 *LiteOS\CMSIS* 目录下的 *cmsis_liteos.c*。

在 *LiteOS/Config* 分组下，添加 *LiteOS\Config* 目录下的所有.h 的头文件。

添加完成后各 Group 下的文件如图 2-10 所示。

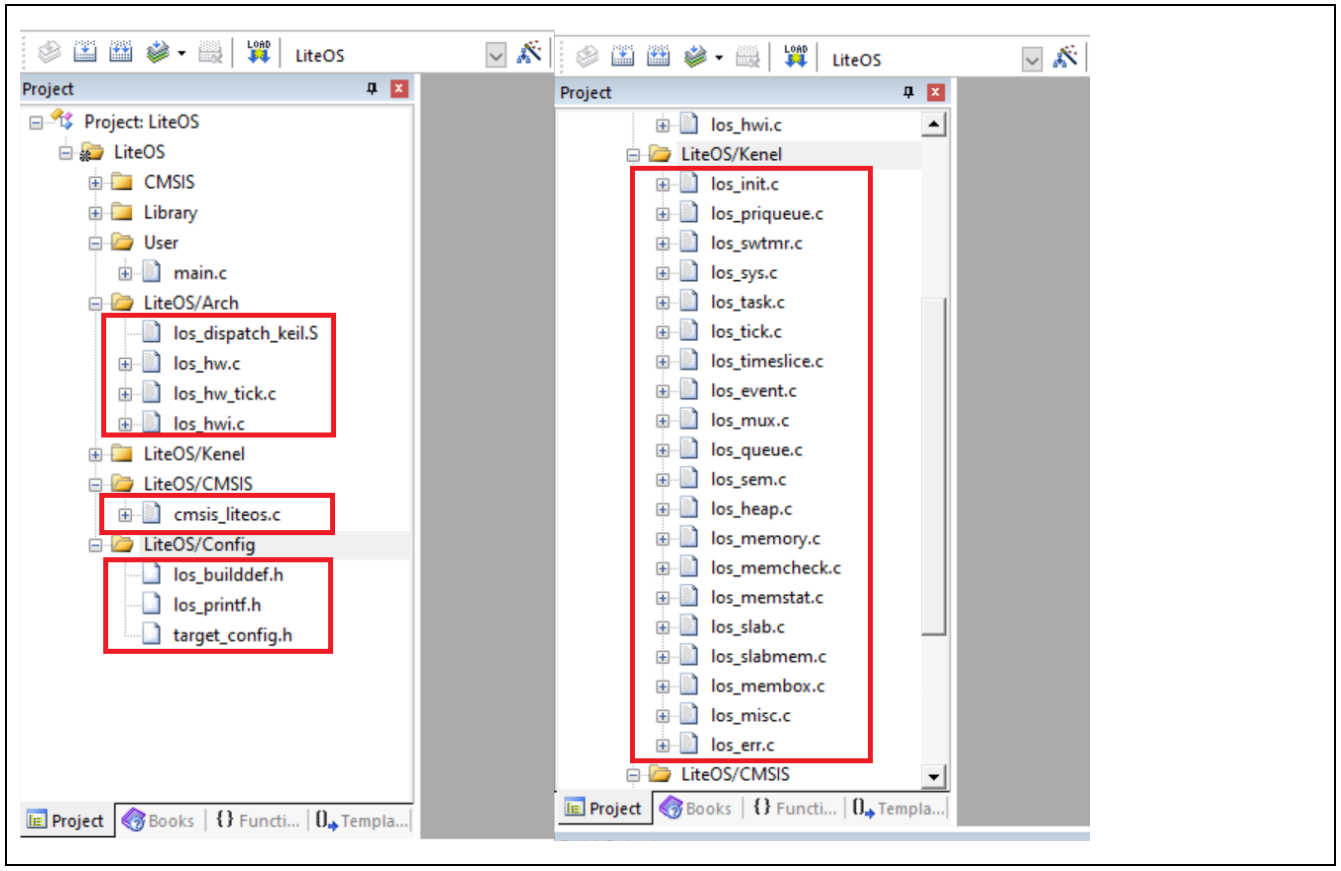

图 2-10 添加的 LiteOS 文件

2.4 修改 Keil Option

修改 Target 选项，如图 2-11 所示。

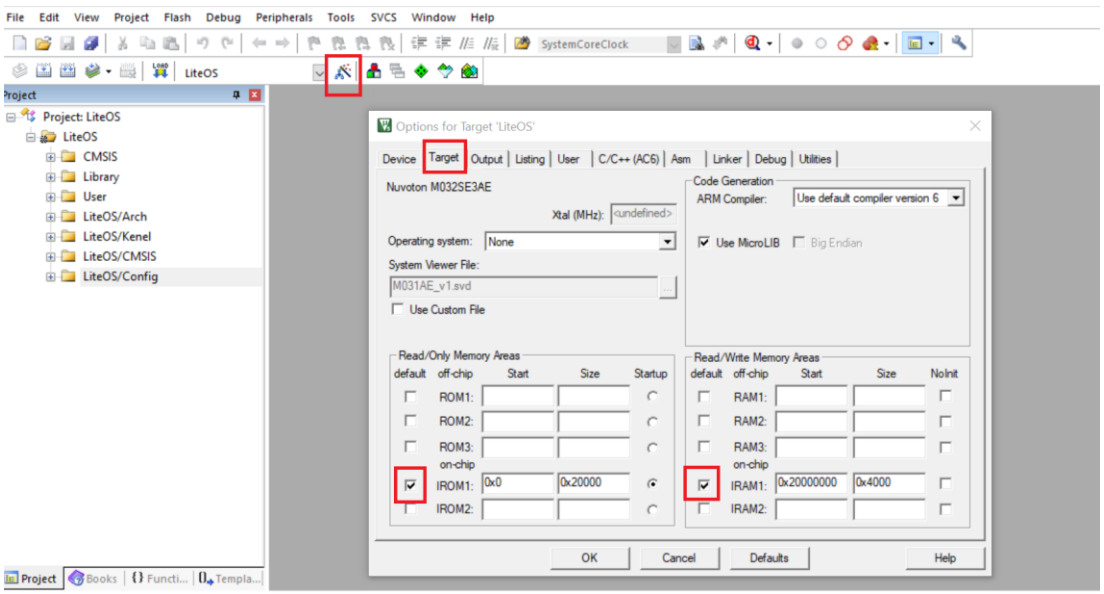

图 2-11 修改 Keil Option->Target

修改 Linker 选项，如图 2-12 所示。

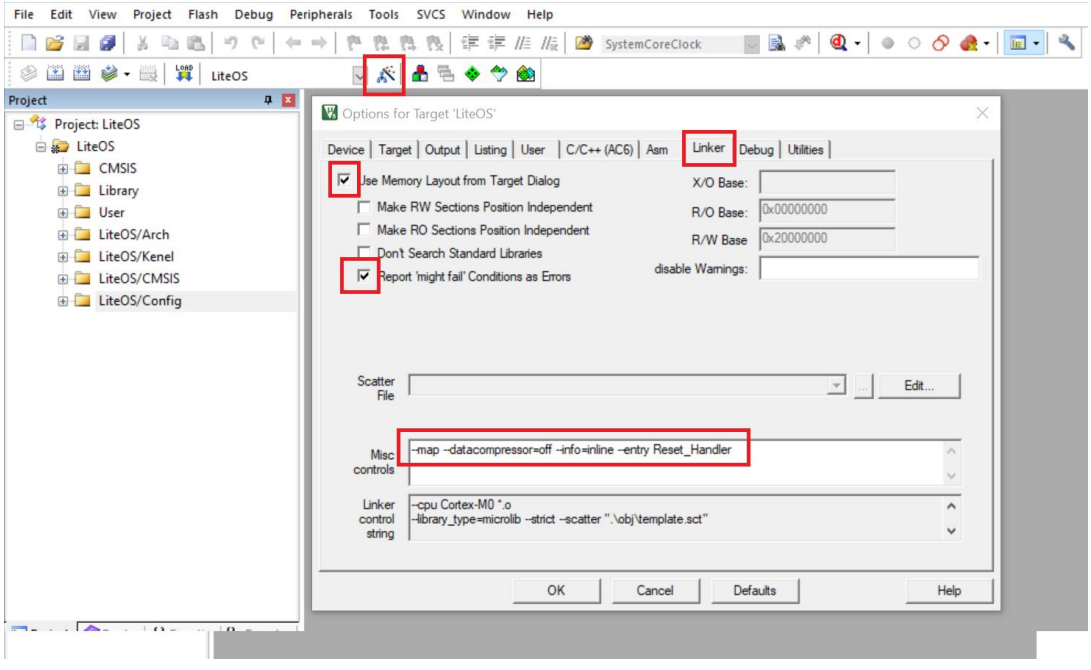

图 2-12 修改 Keil Option->Linker

修改 Output 选项，如图 2-13 所示。

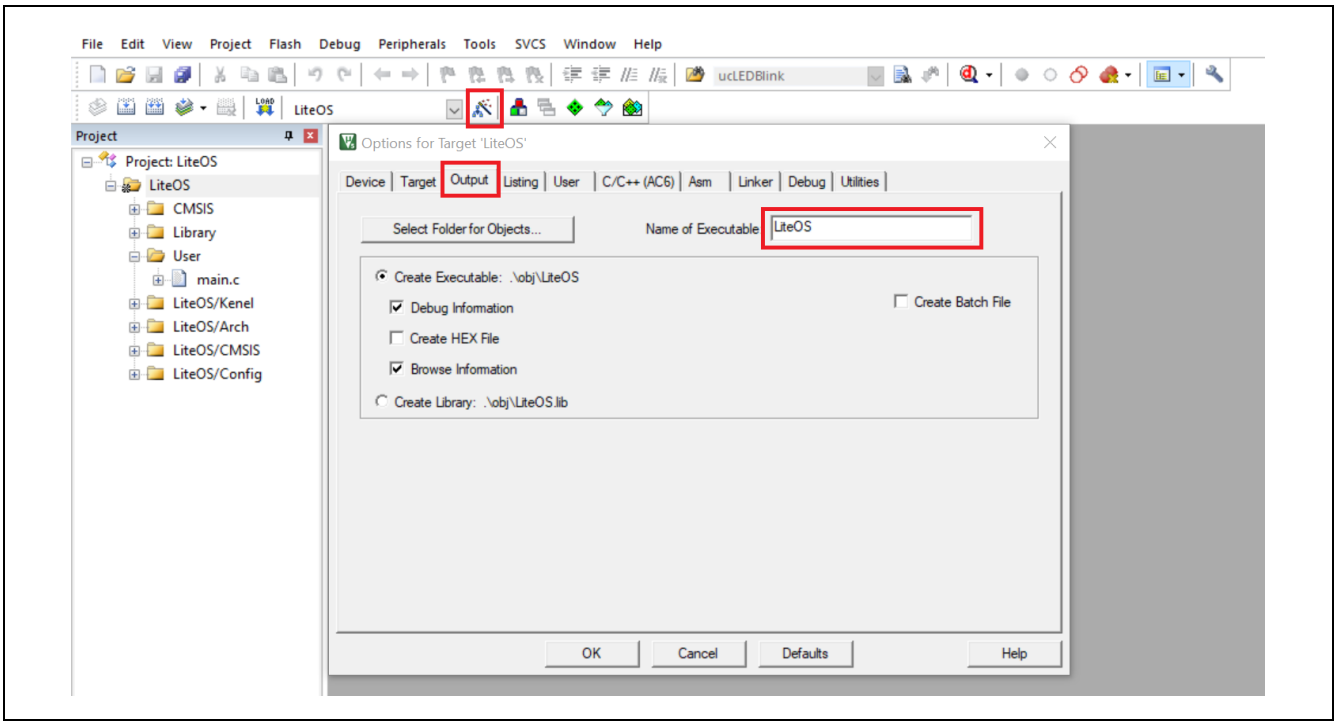

图 2-13 修改 Keil Option->Output

添加头文件目录，如图 2-14所示。

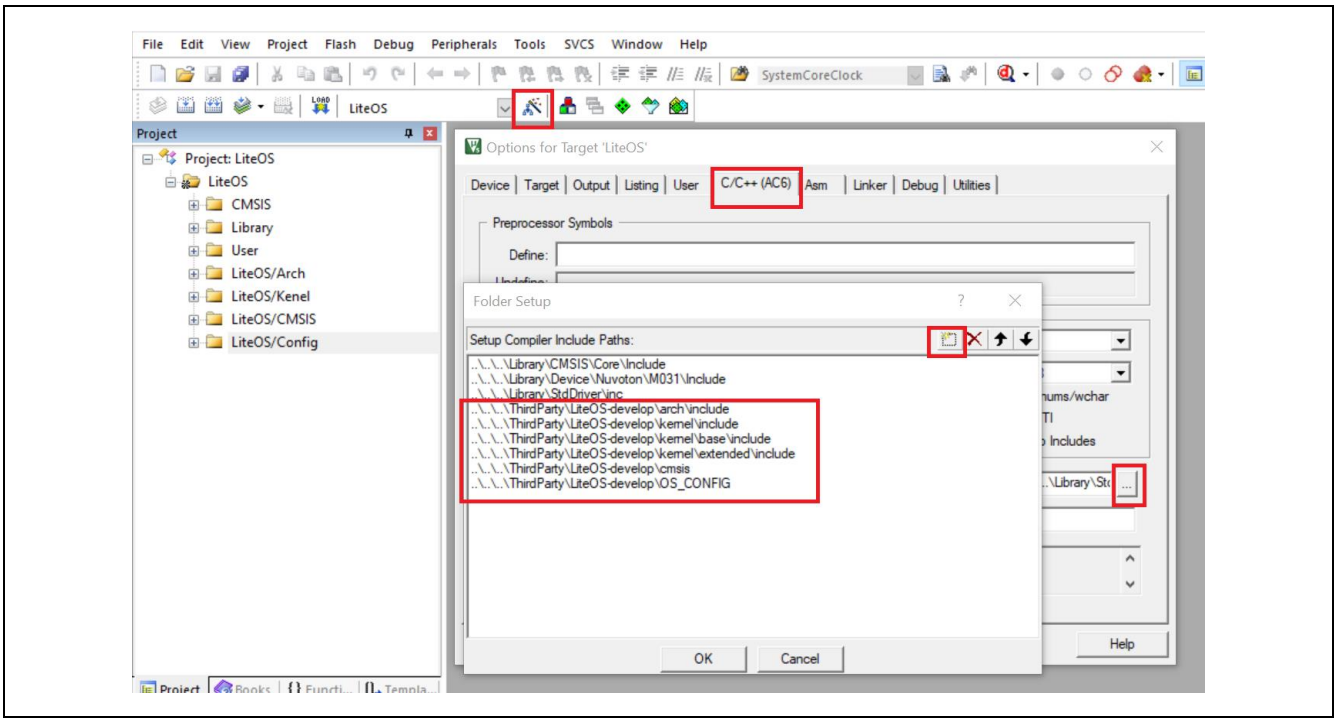

图 2-14 修改头文件路径 Keil Option->C/C++(AC6)

2.5 修改 target_config.h 文件

文件位于*\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\OS_CONFIG* 目录下，修改该文件，新增包含 *NuMicro.h*。

```
#include "NuMicro.h"
#include "los_typedef.h"
#include <stdio.h>
#include <string.h>
```

因为 M031SE3AE SRAM 为 16 KB，还需修改程序中 BOARD_SRAM_SIZE_KB 为 16 KB。

```
/**
 * Without modify default link script and startup file, In order to use all the remaining
 * SRAM space as LiteOS`s heap, User must config the start address and size of the board`s
 * memory, Because in KEIL compiler, the program can`t get the end address of SRAM.
 */
#define BOARD_SRAM_START_ADDR     0x20000000
#define BOARD_SRAM_SIZE_KB        16
#define BOARD_SRAM_END_ADDR       (BOARD_SRAM_START_ADDR + 1024 * BOARD_SRAM_SIZE_KB)
```

匹配 Keil V6 compiler，需修改 heap memory 编译条件。

```
/**
 * Config the start address and size of the LiteOS`s heap memory
 */
#if defined ( __CC_ARM )||((__ARMCC_VERSION) && (__ARMCC_VERSION >= 6000000))
```

2.6 修改 los_typedef.h 文件

文件位于*\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\kernel\include* 目录下，修改该文件，新增包含 *NuMicro.h* 。

```
#ifndef _LOS_TYPEDEF_H
#define _LOS_TYPEDEF_H
#include "NuMicro.h"
#include "los_builddef.h"
```

2.7 修改 los_hw.h 文件

文件位于*\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\arch\include* 目录下， M031 系列芯片没有浮点运算单元，因此文件中定义 FPU_EXIST 的代码需更改，更改后如下。

```
/**
 * @ingroup los_hw
 * FPU exist macro: If it exist, save EXC_RETURN value in the context of the task to check
 * the size of the stack frame(BIT_4: 1 ---> 8 words, 0 ---> 26 words) used by the task.
 */
#define FPU_EXIST  0   // FPU does not exist
```

2.8 修改 main.c 文件

文件位于*\EC_M031_LiteOS_V1.00\SampleCode\ExampleCode\LiteOS* 目录下，修改该文件，新增包含文件 *los_base.h* 、 *los_sys.h* 、*los_typedef.h* 、*los_task.h*。

```
/* ---------------------- LiteOS includes ----------------------------------*/
#include "los_base.h"
#include "los_sys.h"
#include "los_typedef.h"
#include "los_task.h"
```

定义线程相关全部变量。

```
uint32_t g_TskHandle;
```

创建线程 1，线程 1 函数中每隔 1000 ms 执行一次并打印“This is task1,count is ***”。

```
/*---------------------------------------------------------------------------*/
/* Print message per 1000ms                                                  */
/*---------------------------------------------------------------------------*/
void task1(void)
{
    uint32_t count = 1;

    while (1)
    {
        printf("This is task1,count is %d\r\n", count++);
        LOS_TaskDelay(1000);
    }
}

/*---------------------------------------------------------------------------*/
/* Creat LiteOS task1                                                        */
/*---------------------------------------------------------------------------*/
uint32_t creat_task1(void)
{
    uint32_t uwRet = LOS_OK;
    TSK_INIT_PARAM_S task_init_param;

    task_init_param.usTaskPrio = 0;                       /**< Task priority         */
    task_init_param.pcName = "task1";                     /**< Task name             */
    task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)task1; /**< Task entrance function*/
    task_init_param.uwStackSize = 0x200;                  /**< Task stack size       */

    uwRet = LOS_TaskCreate(&g_TskHandle, &task_init_param);

    if (LOS_OK != uwRet)
    {
        return uwRet;
    }

    return uwRet;
}
```

创建线程 2，线程 2 函数中每隔 2000 ms 执行一次并打印“This is task2,count is ***”。

```
/*---------------------------------------------------------------------------------------*/
/* Print message per 2000ms                                                              */
/*---------------------------------------------------------------------------------------*/
void task2(void)
{
    uint32_t count = 1;

    while (1)
    {
        printf("This is task2,count is %d\r\n", count++);
        LOS_TaskDelay(2000);
    }
}

/*---------------------------------------------------------------------------------------*/
/* Creat LiteOS task2                                                                    */
/*---------------------------------------------------------------------------------------*/
uint32_t creat_task2(void)
{
    uint32_t uwRet = LOS_OK;
    TSK_INIT_PARAM_S task_init_param;

    task_init_param.usTaskPrio = 1;                       /**< Task priority         */
    task_init_param.pcName = "task2";                     /**< Task name             */
    task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)task2; /**< Task entrance function*/
    task_init_param.uwStackSize = 0x200;                  /**< Task stack size       */

    uwRet = LOS_TaskCreate(&g_TskHandle, &task_init_param);

    if (LOS_OK != uwRet)
    {
        return uwRet;
    }

    return uwRet;
}
```

主函数 main 重新设计。

```
/*
 * This is a LiteOS project for M031 series MCU. Users could based on this project to
 * create their own application without worry about the Keil project settings.
 * This project shows how to runs LiteOS on M031.
 */
int main(void)
{
    uint32_t uwRet = 0;
    SYS_Init();

    /* Init UART0 to 115200-8n1 for print message */
    UART_Open(UART0, 115200);

    /* Connect UART to PC, and open a terminal tool to receive following message */
    printf("Hello World\n");
    printf("\n\nCPU @ %dHz\n", SystemCoreClock);
    printf("+-----------------------------------------------+\n");
    printf("|    Lite OS Sample Code     |\n");
    printf("+-----------------------------------------------+\n\n");
    /* Init LiteOS Kernel */
    uwRet = LOS_KernelInit();
```

```
    if (uwRet != LOS_OK)
    {
        return LOS_NOK;
    }

    /* Creat task1 */
    uwRet = creat_task1();

    if (uwRet != LOS_OK)
    {
        return LOS_NOK;
    }

    /* Creat task2 */
    uwRet = creat_task2();

    if (uwRet != LOS_OK)
    {
        return LOS_NOK;
    }

    /* Start LiteOS */
    LOS_Start();

    /* Got no where to go, just loop forever */
    while (1);
}
```

2.9 修改各目录名称

对各目录名称进行相应修改，并删除非必要文件，如图 2-15 所示。

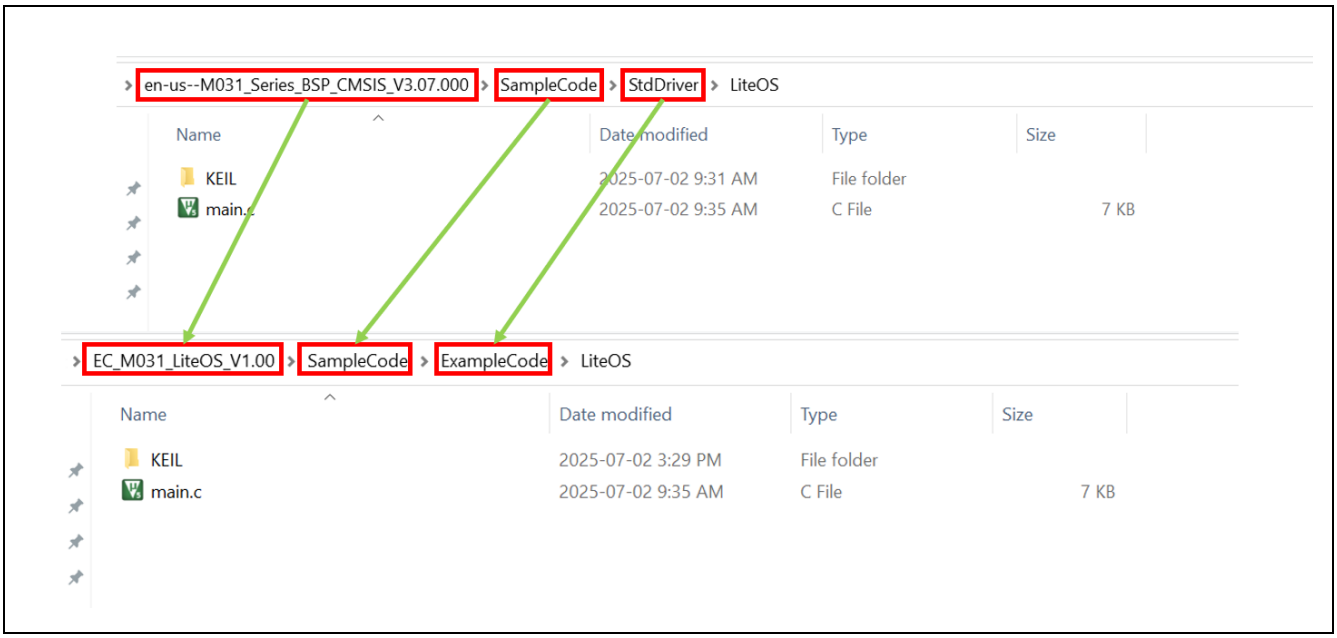

图 2-15 修改各目录名称

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M031_Series_BSP_CMSIS_V3.07.000
- IDE 版本
  - Keil uVersion 5.41

3.2 硬件需求

- 电路组件
  - NuMaker-M031SE3AE V1.1
- 线路示意图
  - 通过设置 Nu-Link2-Me 上的拨码开关，使能其 USB 转串口功能。

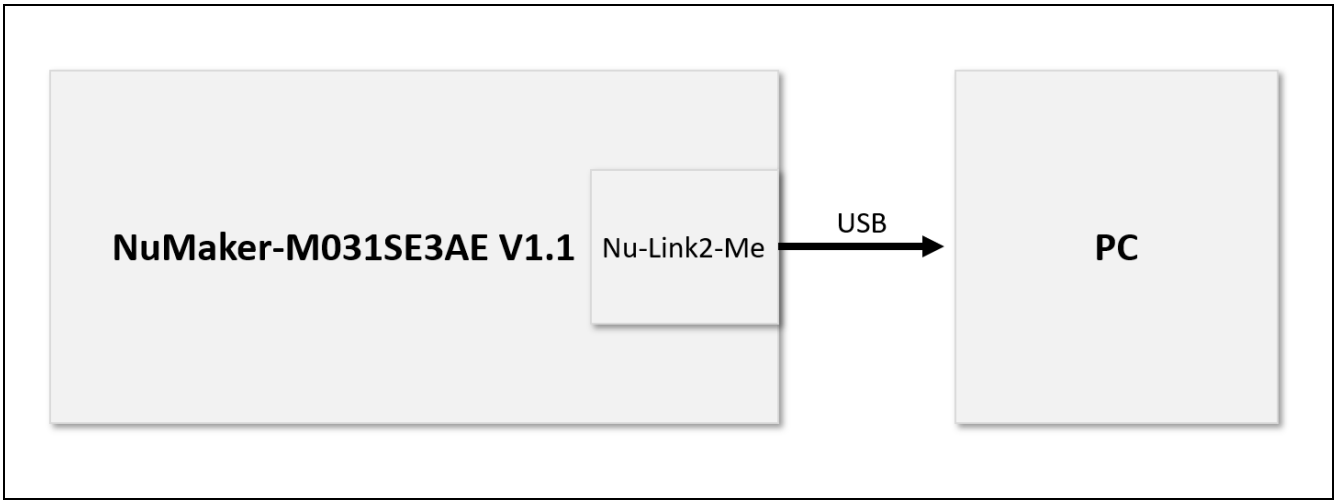

图 3-1 线路示意图

4. 目录信息

目录结构如图 4-1 目录信息所示。

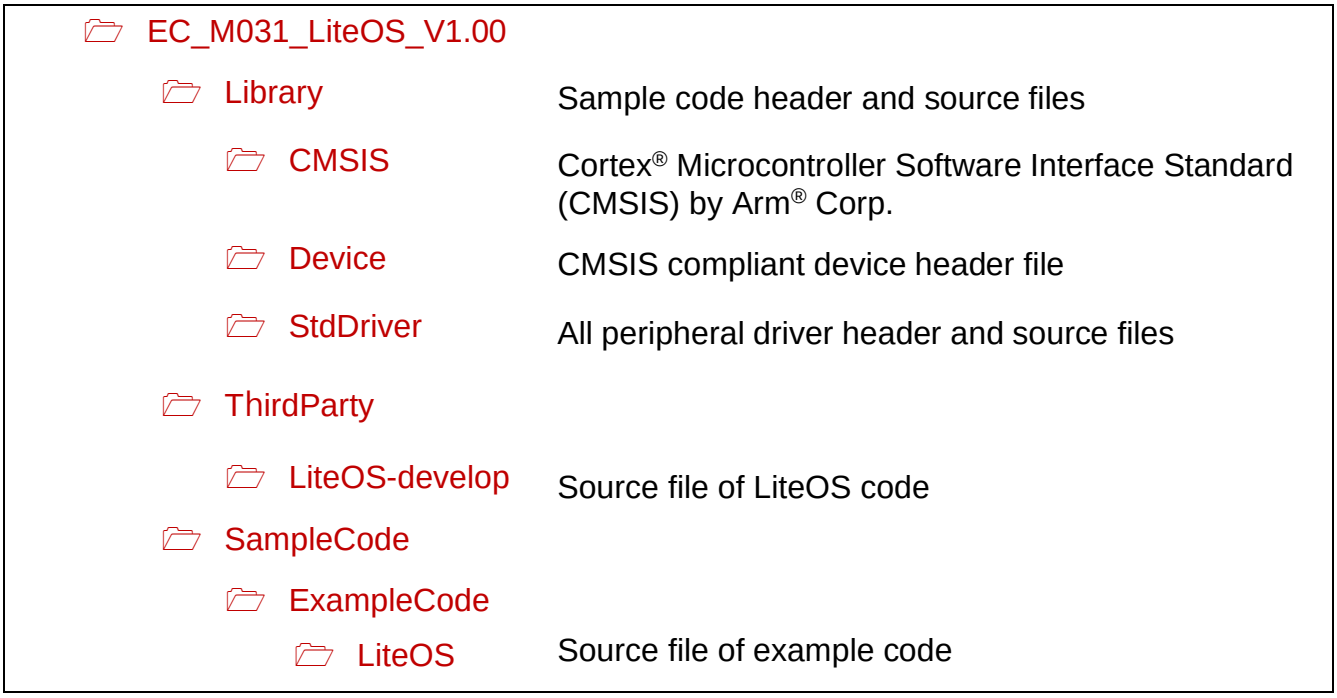

图 4-1 目录信息

5. 范例程序执行

根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *LiteOS.uvprojx*。

1. 进入编译模式界面

   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式

2. 进入仿真模式界面

   - 执行代码

6. 修订纪录

Date	Revision	Description
2025.07.24	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.